On-line Algorithms for Path Selection in a Nonblocking Network

(Extended Abstract)

Sanjeev Arora[1,2] *Tom Leighton*[1,2] *Bruce Maggs*[2]

[1]Mathematics Department and
[2]Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

Nonblocking networks arise in a variety of applications involving communications. The most well known examples include telephone networks, data networks, and distributed memory architectures. Although asymptotically optimal constructions are known for nonblocking networks in a variety of models, it is generally not known how to select paths for the desired network connections efficiently on-line. In this paper, we present the first optimal-time algorithms for path selection in an optimal-size nonblocking network. In particular, we describe a bounded-degree, $O(N \log N)$-switch nonblocking network that can realize any sequence of connections and disconnections among N terminals with $O(\log N)$ bit-step delay. Viewed in the context of a telephone switching network, our network and algorithm can handle any sequence of calls among N parties with $O(\log N)$ bit-step delay per call (even if many calls are made at once). Parties can hang up and call again whenever they like, and multiparty calls can be made without affecting the performance of the algorithm — every call is still put through in $O(\log N)$ time. Viewed in the context of distributed memories for parallel machines, our algorithm allows any processor to access any idle block of memory within $O(\log N)$ bit-steps at any time — no matter what other connections have been made previously or are being made simultaneously.

1 Introduction

1.1 Definitions

Nonblocking networks arise in a variety of communications applications. Common examples include telephone networks, data networks, and network architectures for parallel machines. In a typical application, there are $2N$ *terminals* (usually thought of as N *inputs* and N *outputs*) interconnected by switches (also called nodes) that can be set so as to link the inputs to the outputs with node-disjoint paths according to a specified permutation. The goal is to interconnect the terminals and switches so that any unused input–output pair can be connected by a path of unused switches, no matter what other paths exist at the time. Such a network is said to be *nonblocking*. For example, the 6-terminal graph shown in Figure 1 is nonblocking since no matter which input–output pairs are connected by a path, there is a node-disjoint path linking any unused input–output pair. In particular, if Bob is talking to Alice and Ted is talking to Carol, then Pat can still call Vanna.

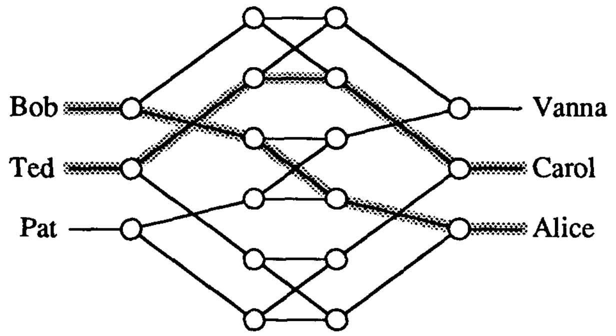

Figure 1: A nonblocking network with 3 inputs and 3 outputs.

The notion of a nonblocking network has several variations. The nonblocking network in Figure 1 is an example of the most commonly studied type. This network is called a *strict-sense nonblocking connector*, because no matter what paths are established in the network, it is possible to establish a path from any unused input to any unused output. A slightly weaker notion is that of a *wide-sense nonblocking connector*. A wide-sense nonblocking connector does not make the same guarantee as a strict-sense nonblocking connector. A network is a wide-sense nonblocking connector if there is an algorithm for establishing paths in the network one after another so that after each path is established, it is still possible to connect any unused input to any

This research was supported by the Defense Advanced Research Projects Agency under Contracts N00014-87-K-825 and N00014-89-J-1988, the Air Force under Contract AFOSR-89-0271, and the Army under Contract DAAL-03-86-K-0171.

unused output. Still weaker is the notion of a *rearrangeable connector.* A rearrangeable connector is capable of realizing any 1–1 connection of inputs to outputs with node-disjoint paths provided that all of the connections to be made are known in advance. A nonblocking or rearrangeable connector is a called a *generalized connector* if it has the additional property that each input can be simultaneously connected to an arbitrary set of outputs, provided that every output is connected to just one input. Generalized connectors are useful for multiparty calling in a telephone network as well as for broadcasting in a parallel machine.

1.2 Previous work

Nonblocking and rearrangeable networks have a rich and lengthly history. See [22] for an excellent survey and [8, 9] for more comprehensive descriptions of previous results. In 1950, Shannon [25] proved that any rearrangeable or nonblocking connector with N-inputs and N-outputs must have $\Omega(N \log N)$ edges. Further work on lower bounds can be found in [4, 10, 23, 24]. In 1953, Clos constructed a strict-sense nonblocking connector with $O(N^{1+1/j})$ edges and depth j, for fixed j. (The degree is not bounded). Bounded depth nonblocking networks have subsequently been studied extensively [7, 9, 16, 17, 21, 24]. In 1964, Benes [5] discovered a bounded-degree rearrangeable connector, now called a Benes network, with depth $O(\log N)$ and size $O(N \log N)$. Ofman [18] followed with a generalized rearrangeable connector of size $O(N \log N)$. Cantor [6] discovered a bounded-degree $O(\log N)$-depth strict-sense nonblocking connector with $O(N \log^2 N)$ edges. The existence of a bounded-degree strict-sense nonblocking connector with size $O(N \log N)$ and depth $O(\log N)$ was first proved by Bassalygo and Pinsker [3]. Although the Bassalygo and Pinsker proof is nonconstructive, subsequent work on the explicit construction of expanders [15] yielded a construction.

More recent work has focused on the construction of generalized nonblocking connectors. In 1973, Pippenger [20] constructed a wide-sense generalized nonblocking connector with $O(N \log^2 N)$ edges. This result was later improved to $O(N \log N)$ edges by Feldman, Friedman, and Pippenger [9]. Recently Turner showed how to construct generalized nonblocking networks by cascading two Clos or Cantor networks [26]. In these networks, all of the parties in a multiparty call must be known at the time that the call is made.

Unfortunately, there has not been as much progress on the problem of setting the switches so as to realize the connection paths. Indeed, many of the references cited previously show that there exists a way of setting the switches so as to realize the desired paths, but are unable to provide any reasonable algorithms (on-line or off-line) for actually finding the right switch settings. There are some exceptions. For example, it is possible to find paths on-line in $O(\log N)$ time for the naive $\Theta(N^2)$-switch nonblocking networks (e.g. an $N \times N$ mesh of trees [12]), and in polylogarithmic time for some of the $\Theta(N^{1+\epsilon})$-switch constructions mentioned previously. More recently, Lin [19] found polylogarithmic time path selection algorithms for $O(N \log^2 N)$-switch networks. No fast algorithms were known for the $O(N \log N)$-switch networks, however, and no $O(\log N)$-step algorithms were known for any of the $o(N^{1+\epsilon})$-switch networks.

1.3 Our results

In this paper, we describe an $O(N \log N)$-switch nonblocking network for which each path connection can be made on-line in $O(\log N)$ bit-steps. Moreover, the network can realize any multiparty call in $O(\log N)$ word-steps on-line, provided that all the parties to each call are known at the time of the call. (If all of the parties are not known in advance, then there is a cost of $O(\log N)$ steps for each group of callers added later on.) The algorithms work even if many calls are made at once — every call still gets through in $O(\log N)$ bit-steps, no matter what calls were made previously and no matter what calls are currently active, provided that no two inputs try to access the same output at the same time. (If many inputs inadvertently try to access the same output at the same time, all but one of the inputs will receive a busy signal. The busy signals are also returned in $O(\log N)$ bit-steps, but, at present, we require the use of a sorting circuit [2] to generate the busy signals. Alternatively, we could merge the calling parties together, but this also requires the use of a sorting circuit.) In all scenarios, the size of the network and the speed of the path selection algorithm are asymptotically optimal.

In addition to providing the first optimal solution to the abstract telephone switching problem, our results significantly improve upon previously known algorithms for bit-serial (or byte-serial) packet routing. Previously, $O(\log N)$-bit-step algorithms for packet routing were known only in the special case where are all packet paths are created or destroyed at the same time, and even then only by resorting to the AKS sorting circuit [2], or by using randomness on the hypercube [1]. In many circuit-switched parallel machines, however, packets are of varying lengths and packet paths are created and destroyed at arbitrary times, thereby requiring that paths be routed in a nonblocking fashion – which is something that previously discovered algorithms were not capable of doing. Even without worrying about the nonblocking property, our results provide the first non-AKS $O(\log N)$-bit-step algorithms for bit-serial packet routing on a bounded-degree network. Although we have not yet tested our algorithms experimentally, we are optimistic that they will perform well in practice.

1.4 Our approach

The family of networks that we use to obtain these results combines expanders and the Benes network in a manner similar to the multibutterfly network described by Upfal [27] and the nonblocking networks of Bassalygo and Pinsker [3]. We refer to the networks as ***multi-Benes networks***. The details of the construction are provided in Section 2 of the paper. We can also apply our results to bandwidth-limited switching networks such as fat-trees [14], and obtain optimal performance in terms of load factor. Such results may be more useful in the context of telephone networks, where there are limitations on the number of calls based on the proximity of the calls (e.g., it is not possible for everyone on the East Coast to call everyone on the West Coast at the same time).

The description and analysis of the path selection algorithms is divided into three sections. In Section 3, we describe on-line algorithms for adding a single connection path in the network. These algorithms are similar to the fault-tolerant routing algorithms in [13]. Indeed, we can think of currently-used wires as being faulty since they cannot be used to form a new connection path. Similarly, the algorithms we describe for routing in nonblocking networks can easily be extended to be highly tolerant to faults in the network [11].

In Section 4, we describe an $O(\log N)$-bit-step algorithm for bit-serial routing in a multibutterfly. This algorithm relies on a unique-neighbor property possessed by all highly-expanding graphs. By implementing this algorithm on the multi-Benes network and combining it with the methods of Section 3, we produce an algorithm that can handle many calls at the same time, independent of what calls have been made previously and what calls are currently connected.

In Section 5, we describe algorithms for handling multiparty calls, and situations where many inputs try to reach the same output simultaneously. Some of these algorithms rely on sorting circuits and are not as practical as those described in Sections 3 and 4.

1.5 Remarks and conventions

To simplify the explanation of the algorithms and results in this paper, we have adopted some conventions that may differ from the way that this material is treated in the more applied literature. For example, we always route paths in a node-disjoint fashion. In practice, this is wasteful of resources, and it is desirable to route paths in an edge-disjoint manner. All of our definitions and results can be applied in this setting as well.

When packets are routed in an edge-disjoint fashion, it is possible to have more inputs and outputs for the same size network. For example, the butterfly shown in Figure 2 could have 16 inputs (2 per first level switch)

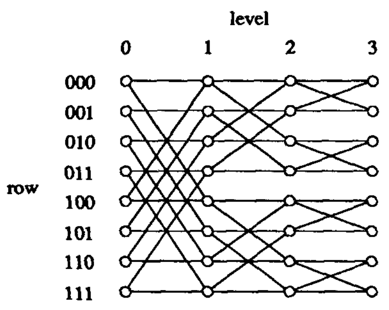

Figure 2: An 8-input butterfly network.

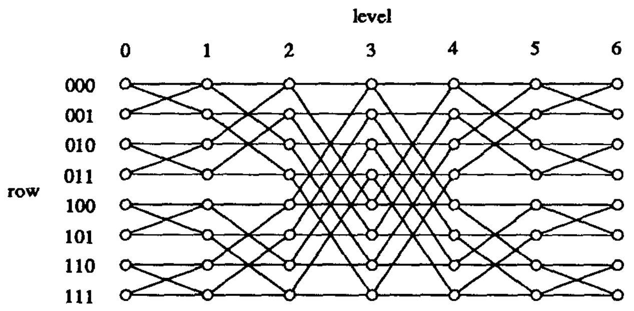

Figure 3: An 8-input Benes network.

and the multibutterfly shown in Figure 4 could have 32 inputs (4 per first level switch).

It is also worth noting that all of the results in this paper hold for a much broader class of networks than multibutterflies and multi-Benes networks. In particular, each basic butterfly component used to make a multibutterfly or multi-Benes network can be replaced by any Delta network. A *Delta* network is a regular network formed by splitters like the butterfly, but for which the individual connections within each splitter can be arbitrary.

2 The multi-Benes and multibutterfly networks

Our nonblocking network is constructed from a Benes network and expander graphs in much the same way as a multibutterfly network is constructed from a butterfly network. We start by describing the butterfly, Benes, and multibutterfly networks.

An N-input butterfly has $\lg N + 1$ levels, each with N-nodes. An example is shown in Figure 2. The Benes network is a $(2 \lg N+1)$-level network consisting of back-to-back butterflies. For example, see Figure 3.

A multibutterfly is formed by gluing together butterflies in a somewhat unusual way. In particular, given 2 N-input butterflies G_1 and G_2 and a collection of permutations $\Pi = \langle \pi_0, \pi_1, \ldots, \pi_{\lg N} \rangle$ where $\pi_l : [0, \frac{N}{2^l}-1] \rightarrow [0, \frac{N}{2^l} - 1]$, a 2-butterfly is formed by merging the node

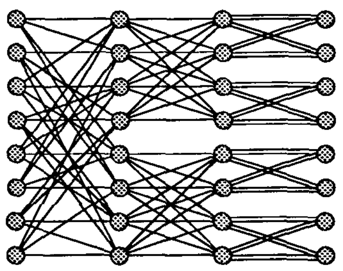

Figure 4: An 8-input 2-butterfly network.

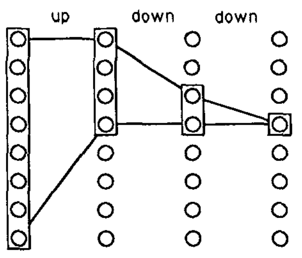

Figure 5: The logical path from an input to output 011.

in row $\frac{jN}{2^l} + i$ of level l of G_1 with the node in row $\frac{jN}{2^l} + \pi_l(i)$ of level l of G_2 for all $0 \le i \le \frac{N}{2^l} - 1$, all $0 \le j \le 2^l - 1$, and all $0 \le l \le \lg N$. The result is an N-input $(\lg N + 1)$-level graph in which each node has 4 inputs and 4 outputs. Of the 4 output edges at a node, two are *up* outputs and two are *down* outputs (with one up edge and one down edge coming from each butterfly). Multibutterflies (i.e., d-butterflies) are composed from d butterflies in a similar fashion using $d-1$ sets of permutations, $\Pi^{(1)}, \ldots, \Pi^{(d-1)}$, resulting in a $(\lg N + 1)$ level network with $2d \times 2d$ switches. For example, see Figure 4.

The notion of up and down edges can be formalized in terms of splitters. More precisely, the edges from level l to level $l+1$ in rows $\frac{jN}{2^l}$ to $\frac{(j+1)N}{2^l} - 1$ in a multibutterfly form a *splitter* for all $0 \le l < \lg N$ and $0 \le j \le 2^l - 1$. Each of the 2^l splitters starting at level l has $\frac{N}{2^l}$ inputs and outputs. The outputs on level $l+1$ are naturally divided into $\frac{N}{2^{(l+1)}}$ *up* outputs and $\frac{N}{2^{(l+1)}}$ *down* outputs. By definition, all splitters on the same level l are isomorphic, and each input is connected to d up outputs and d down outputs according to the butterfly and the permutations $\pi_l^{(1)}, \ldots, \pi_l^{(d-1)}$. Hence, any input and output of the multibutterfly are connected by a single logical (up-down) path through the multibutterfly, but each step of the logical path can be taken on any one of d edges. For example, see Figure 5.

The most important characteristic of a multibutterfly

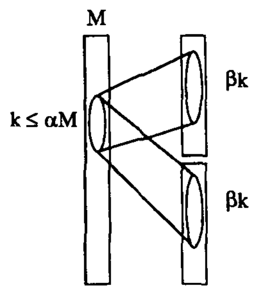

Figure 6: A splitter with expansion property (α, β).

is the set of permutations $\Pi^{(1)}, \ldots, \Pi^{(d-1)}$ that prescribe the way in which the component butterflies are to be merged. For example, if all of the permutations are the identity map, then the result is the *dilated butterfly* (i.e., a butterfly with d copies of each edge). We are most interested in multibutterflies that have expansion properties. In particular, we say that an M-input splitter has *expansion property* (α, β) if every set of $k \le \alpha M$ inputs is connected to at least βk up outputs and βk down outputs for $\beta > 1$. Similarly, we say that a multibutterfly has *expansion property* (α, β) if each of its component splitters has expansion property (α, β). For example, see Figure ??.

If the permutations $\Pi^{(1)}, \ldots, \Pi^{(d-1)}$ are chosen randomly, then with good probability the resulting d-butterfly has expansion property (α, β) for any d, α, and β for which $2\alpha\beta < 1$ and

$$d \quad < \quad \beta + 1 + \frac{\beta + 1 + \ln 2\beta}{\ln(\frac{1}{2\alpha\beta})}. \qquad (1)$$

Constructions for splitters and multibutterflies with good expansion properties are known although the expansion properties are generally not as good as those obtained from randomly-generated graphs.

Like a multibutterfly, a multi-Benes network is formed from Benes networks by merging them together. A 2-multi-Benes network is shown in Figure 7. An N-input multi-Benes network has $2 \lg N + 1$ levels labeled 0 through $2 \lg N$. Levels $\lg N$ through $2 \lg N$ form a multibutterfly, while levels 0 through $\lg N$ form the mirror image of a multibutterfly.

As in the multibutterfly, the edges in levels $\lg N$ through $2 \lg N$ are partitioned into splitters. Between levels 0 and $\lg N$, however, the edges are partitioned into *mergers*. More precisely, the edges from level l to level $l+1$ in rows $j2^{l+1}$ to $(j+1)2^{l+1} - 1$ form a *merger* for all $0 \le l < \lg N$ and $0 \le j \le N/2^{l+1} - 1$. Each of the $N/2^{l+1}$ mergers starting at level l has 2^{l+1} inputs and outputs. The inputs on level l are naturally divided into 2^l *up* inputs and 2^l *down* inputs. All mergers on the same level l are isomorphic, and each input is connected to $2d$ outputs. There is a single (trivial) logical

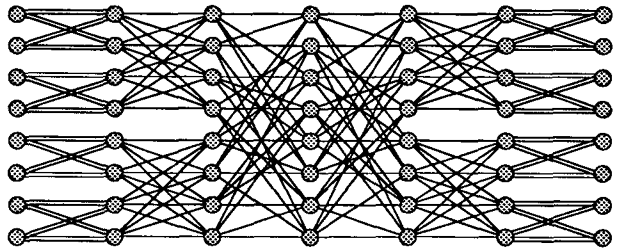

Figure 7: An 8-input 2-multi-Benes network.

path from any input of a multi-Benes network through the mergers on the first $\lg N$ levels to the single splitter on level $\lg N$. From level $\lg N$ there is a single logical path through the splitters to any output. In both cases, the logical path can be realized by many physical paths.

We say that an M-output merger has expansion property (α, β) if every set of $k \leq \alpha M$ inputs (up or down) is connected to at least $2\beta k$ outputs $\beta > 1$. With nonzero probability, a random set of permutations yields a merger with expansion property (α, β) for any d, α, and β for which $\alpha\beta < 1$ and

$$2d \quad < \quad 2\beta + 1 + \frac{2\beta + 1 + \ln 2\beta}{\ln(\frac{1}{2\alpha\beta})}. \qquad (2)$$

We say that a multi-Benes network has expansion property (α, β) if each of its component mergers and splitters has expansion property (α, β). The multibutterflies and multi-Benes networks considered throughout this paper are assumed to have expansion property (α, β).

3 A non-blocking path selection algorithm for the multi-Benes network

In this section we describe an efficient on-line algorithm for satisfying connection requests in a multi-Benes network. The algorithm establishes a path from an unused input to an unused output in $O(\log N)$ bit-steps, where N is the number of rows. Throughout, we assume that at most one input tries to access any output at a time, and that each input accesses at most one output at a time. Algorithms for multiparty calling are deferred to Section 5.

In order for the algorithm to succeed, the multi-Benes network must be lightly loaded by some fixed constant factor L. Thus, in an N-row multi-Benes network, we only make connections between the N/L inputs and outputs in rows that are multiples of L. Since the other inputs and outputs are not used, the first and last $\lg L$ levels of the network can be removed, and the N/L inputs and outputs can each be connected directly to their L descendants and ancestors on levels $\lg L$ and $2\lg N - \lg L$, respectively.

The basic idea is to treat the switches through which paths have already been established as if they were faulty and to apply the fault propagation techniques from [13] to the network. In particular, we define a node to be *busy* if there is a path currently routing through it, and we recursively define a node to be *blocked* if all of its up outputs or all of its down outputs are busy or blocked. More precisely, switches are declared to be blocked according to the following rule. Working backwards from level $2\lg N - \lg L - 1$ to level $\lg N$, a switch is declared blocked if either all d of its up edges or all d of its down edges lead to busy or blocked switches. From level $\lg N - 1$ to level $\lg L$, a switch is declared blocked if all $2d$ of its outputs lead to busy or blocked switches. A switch that is neither busy nor blocked is said to be *working*.

The following pair of lemmas bound the fraction of input switches that are blocked in every splitter and merger.

Lemma 1 *For $L > 1/2\alpha(\beta - 1)$, at most a 2α fraction of the inputs in any splitter are declared to be blocked. Furthermore, at most an α fraction of the switches are blocked because of busy and blocked switches from the upper outputs, and at most an α fraction are blocked because of the lower outputs.*

Proof: The proof is by induction on level number, starting at level $2\lg N - \lg L$ and working backwards to level $\lg N$. The base case is trivial since there are no blocked switches on level $2\lg N - \lg L$. Suppose the inputs of an M-input splitter contain more than αM switches that are blocked because of the upper (say) outputs. Consider the set U of busy or blocked upper outputs. Since all of the edges out of a blocked input lead to busy or blocked outputs, we can conclude that $|U| \geq \alpha\beta M$. Since all paths passing through the upper outputs must lead to one of $M/2L$ terminals, there can be at most $M/2L$ busy switches among the upper outputs of the splitter. Furthermore, by induction there are at most αM blocked switches among the upper outputs. Thus, $|U| \leq \alpha M + M/2L$. For $L > 1/2\alpha(\beta - 1)$ we have a contradiction. Hence, at most an α fraction of the switches are blocked or busy, as claimed. □

Lemma 2 *For $L > 1/2\alpha(\beta - 1)$, at most a 2α fraction of the upper inputs and a 2α fraction of the lower inputs in any merger are blocked.*

Proof: The proof is like that of Lemma 1 □

After the fault propagation process, every working switch in the first half of the network has an output that leads to a working switch, and every working switch in the second half has both an up output and a down output that lead to working switches. Furthermore, since at most a 2α fraction of the switches in each merger on level $\lg L$ are blocked, each of the N/L inputs has an edge to a working switch on level L. At the other end, each of the N/L outputs can be reached by a working switch on level $2\lg N - \lg L$. As a consequence, we can establish a path through working switches from any unused input to any unused output in $O(\log N)$ bit-steps using a simple greedy algorithm. Since the declaration of blocked switches takes just $O(\log N)$ bit-steps, and since the greedy routing algorithm is easily accomplished in $O(\log N)$ bit-steps, the entire process takes just $O(\log N)$ bit-steps.

4 Establishing many paths at once

The preceding algorithm can establish any single additional path successfully in $O(\log N)$ bit-steps. While this is sufficient to show that the multi-Benes network is nonblocking, it is not sufficient to handle many calls at once. In what follows, we describe an algorithm for routing an arbitrary number of additional calls in $O(\log N)$ bit-steps. As before, we assume for the time being that each input and output is involved in a at most one two-party call. Extensions to the algorithm for handling multiparty calls are described in Section 5. We also assume that paths are established between inputs and outputs on rows congruent to 0 mod L in the multi-Benes network, where $L \geq 1/\alpha$. This will insure that no splitter or merger is ever overloaded.

To simplify the exposition of the algorithm, we start by describing an algorithm for routing any initial set of paths in a multibutterfly (i.e., we don't worry about the nonblocking aspect of the problem for the time being). This comprises the first known circuit-switching algorithm for the multibutterfly. (Previous routing algorithms for the multibutterfly [13, 27] only worked for the store-and-forward model of routing.) We then modify the definitions of busy and blocked from Section 3 and show how to implement the algorithm in a nonblocking fashion on the multi-Benes network.

4.1 Circuit-switching on a multibutterfly

Our circuit-switching algorithm requires the splitters in the multibutterfly to have a special "unique-neighbors" property defined as follows.

Definition 3 *An M-input splitter is said to have the (α, δ) unique neighbor property if in every subset X of $k \leq \alpha M$ inputs, there are δk nodes in X which have an up-output neighbor that is not adjacent to any other node in X, and there are δk nodes in X which have a down-output neighbor that is not adjacent to any other node in X (i.e., δk nodes in X have a unique up-neighbor, and δk nodes have a unique down-neighbor).*

Lemma 4 *Any splitter with the (α, β) expansion property has the (α, δ) unique-neighbors property where $\delta = 2\beta/d - 1$, provided that $\beta > d/2$.*

Proof: Consider any set X of $k \leq \alpha M$ inputs in an M-input splitter. These nodes have at least βk neighbors among the up (down) outputs. Let n_1 denote the number of these neighbors incident to precisely one node of X, and let n_2 denote the number of neighbors incident to two or more nodes of X. Then $n_1 + n_2 \geq \beta k$ and $n_1 + 2n_2 \leq dk$. Solving for n_1 reveals that $n_1 \geq (2\beta - d)k$. Hence at least $(2\beta/d - 1)k$ of the nodes in X are adjacent to a unique neighbor. □

By Equation 1, we know that randomly generated splitters have the (α, δ) unique-neighbors property where δ approaches 1 as d gets large and α gets small. Explicit constructions of such splitters are not known, however. Nevertheless, we will consider only multibutterflies with the (α, δ) unique-neighbors property for $\delta > 0$ in what follows.

Remark: The (α, β) expansion property ($\beta > d/2$) is a sufficient condition for the unique-neighbors property, but by no means necessary. In fact, we can easily prove the existence of random splitters which have a fairly strong (α, δ) unique-neighbors property for small degree. For such graphs, the routing algorithm we are about to describe is more efficient in terms of hardware required. However, multibutterflies with expansion properties will remain the object of our focus.

It is relatively easy to extend paths from one level to the next in a multibutterfly with the (α, δ) unique-neighbors property. The reason is that those paths at switches with unique neighbors can be trivially extended without worrying about blocking any other path trying to reach the next level. By proceeding recursively, it is easy to see that all the paths can be extended from level l to level $l + 1$ (for any l) in $\log(N/L2^l)/\log(1/1 - \delta)$ steps. In particular, a "step" consists of:

1. every path still waiting to be extended sends out a "proposal" to his output (level $l + 1$) neighbors in the desired direction (up or down),

2. every output node that receives precisely one proposal sends back its acceptance to that proposal,

3. every path receiving an acceptance advances to one of its accepting outputs on level $l + 1$.

Splitters connecting level l to level $l + 1$ have $M = N/2^l$ inputs and at most M/L paths can pass through them by definition of L. Since $L > 1/\alpha$, the set of switches containing paths needing to be extended has size at most αM and we can apply the (α, δ) unique-neighbors property to ensure that at each step, the number of paths still remaining to be extended decreases by a $(1 - \delta)$ factor. Hence, all of the paths are extended in $\log(N/L2^l)/\log(1/(1 - \delta))$ steps, as claimed.

By using the path extension algorithm just described on each level in sequence, we can construct all of the paths in

$$\sum_{l=0}^{\lg N-1} \frac{\log \frac{N}{L2^l}}{\log \frac{1}{1-\delta}} \leq \frac{\log^2 \frac{N}{L2^l}}{2\log \frac{1}{1-\delta}} = O(\log^2 N)$$

bit-steps. To construct the paths in $O(\log N)$ bit-steps we modify this algorithm as follows:

> Given a fraction $< \alpha$ of paths that need to be extended at an M-input splitter, the algorithm does not wait $O(\log M)$ time for every path to be extended before it begins the extension at the next level. Instead, it waits only $O(1)$ steps, in which time the number of unextended paths falls to a fraction ρ of its original value, where $\rho < 1/d$. Now the path extension process can start at the next level. The danger here is that the ρ fraction of paths left behind may find themselves blocked by the time they reach the next level, and so we need to ensure that this won't happen. Therefore, stalled paths send out *place-holders* to all of their neighbors at the next level, and henceforth the neighbors with place-holders participate in path extension at the next level, as if they were paths. Of course, the neighbors holding place-holders must in general extend in both the upper and the lower output portions of the splitter, since they don't know yet which path will ultimately use them.

Notice that a place-holder not only reserves a spot that may be used by a path at a future time, but also helps to chart out the path by continuing to extend ahead. In order to prevent place-holders from multiplying too rapidly and clogging the system — since if the fraction of inputs of a splitter which are trying to extend rises above α, the path extension algorithm ceases to work — we need to ensure that as stalled paths get extended, they send *cancellation signals* to the place-holding nodes ahead of them to tell them they are not needed anymore. When a placeholding node gets cancellations from all the nodes who had requested it to hold their place, it disappears and ceases to extend anymore. It also sends cancellations to any nodes ahead of it that may be holding a place for it. As we shall see, this scheme prevents place-holders from getting too numerous.

The $O(\log N)$-step algorithm for routing paths proceeds in *phases* consisting of the following two types of steps:

- C steps of passing cancellation signals. These cancellation signals travel at the rate of one level per step,
- T steps of extending from one level to the next. In this time, the number of stalled (i.e., unextended) paths at each splitter drops by least a factor of ρ, where $\rho \leq (1-\delta)^T$.

Each path is restricted to extend forward by at most one level during each phase. We refer to the first wave of paths and placeholders to arrive at a level as the *wavefront*. The wavefront moves forward by one level during each phase. If a path or placeholder in the wavefront isn't extended in T steps, it sends placeholders to all of its neighbors at the end of the phase. We will assume that $C \geq 2$ so that cancellation signals have a chance to catch up with the wavefront, and that $d \geq 3$.

The following lemmas will be useful in proving that every path is extended to completion in $\lg N$ phases provided that $L \geq 2/\alpha$ and $\rho < 1/13d$. The key to our approach is to focus on the number of stalled paths (corresponding to real paths *or* placeholders) at the inputs of each splitter. In particular, we let $S(i,t)$ denote the maximum fraction of inputs of any splitter at level i that contain stalled paths at the end of phase t of the algorithm. By definition, $S(i,t) = 0$ for $t < i$, since the wavefront arrives at level i at phase i. Each stalled path generates up to $2d$ placeholders at the next level, which might later become stalled themselves, so it is crucial to keep the number of stalled paths at each level small.

Lemma 5 *If the number of paths (real or placeholder) reaching the inputs of a level i splitter when the wavefront arrives is less than an α fraction of the inputs, then $S(i,t) \leq \rho^{t-i}S(i,i)$ for $t \geq i$.*

Proof: In each phase of the algorithm, the number of stalled paths at the inputs drops by a factor of ρ, provided the number of paths trying to extend is never greater than an α fraction of the inputs of the splitter. Since the number of paths reaching the inputs never increases after the wavefront arrives, this condition is always satisfied. □

Let P_i denote the fraction of inputs containing paths (real and placeholder) in a level i splitter when the wavefront arrives (i.e., at the end of phase $i-1$). Note that if $P_i \leq \alpha$, then $S(i,i) \leq \rho P_i$.

Lemma 6

$$\begin{aligned} P_i \quad \leq \quad & \frac{1}{L} + 2dS(i-1,i-1) \\ & + \sum_{k=2}^{C} 2^k dS(i-k,i-2) \\ & + \sum_{l=1}^{\infty}\sum_{k=1}^{C} 2^{Cl+k} dS(i-Cl-k,i-l-2). \end{aligned}$$

Proof: Note that $1/L$ upperbounds the fraction of real paths that could be in the wavefront, since that is the number of real paths that will ever pass through the splitter. The $2dS(i-1,i-1)$ term represents the fraction of inputs that could have placeholders generated by stalled paths at level $i-1$ (the factor 2 comes in because the number of inputs in a splitter at level $i-1$ is twice as many as those in a level i splitter). Next, $4dS(i-2,i-2)$ upperbounds the fraction of inputs containing placeholders generated by paths stalled at level

$i-2$, but whose placeholders were extended from level $i-1$ to level i. (Note that if $C \geq 3$, for example, we have an $8dS(i-3,i-2)$ contribution instead of an $8dS(i-3,i-3)$ contribution to P_i since paths stalled in level $i-3$ during phase $i-3$, but getting through during phase $i-2$, send a cancellation signal to levels $i-2$, $i-1$, and i during phase $i-1$, and hence they do not contribute placeholders to level i during phase i.) The rest of the terms in the summation may be counted similarly. Finally, though our summation seems to have infinitely many terms, only finitely many of them are non-zero. □

Lemma 7 *The fraction of inputs containing paths (real and placeholders) at any splitter is never more than α provided $L \geq \frac{2}{\alpha}$, $\rho \leq \frac{1}{13d}$, $d \geq 3$, and $C \geq 3$.*

Proof: We prove by induction on i that for $\gamma = \frac{\alpha}{13d}$, $S(i,i) < \gamma$ and $P_i < \alpha$. We only need to prove the inductive step, since the base case is trivial. By the recurrence of Lemma 6, we have:

$$\begin{aligned}
P_i &\leq \frac{1}{L} + 2d\gamma + \sum_{k=2}^{C} 2^k d\gamma\rho^{k-2} \\
&\quad + \sum_{l=1}^{\infty}\sum_{k=1}^{C} 2^{Cl+k} d\gamma\rho^{(C-1)l+k-2} \\
&\leq \frac{1}{L} + 2d\gamma + \frac{4d\gamma(1-(2\rho)^{C-1})}{1-2\rho} \\
&\quad + \frac{d\gamma 2^{C+1}\rho^{C-2}(1-(2\rho)^{C})}{(1-2^{C}\rho^{C-1})(1-2\rho)} \\
&\leq \frac{1}{L} + 2d\gamma + 4.06d\gamma + .3d\gamma.
\end{aligned}$$

Note that we have used the fact that $d \geq 3$, $C \geq 3$, and $\rho \leq 1/13d$. (We really only needed $C \geq 2$, but the constants are better for $C \geq 3$.) Thus if $\gamma = \alpha/13d$ and $L > 2/\alpha$, then $P_i \leq \alpha$. Also, as we noted earlier, $S(i,i) \leq \rho P_i$ and if $\rho < 1/13d$, by Lemma 6, we have: $S(i,i) < \alpha/13d = \gamma$, thereby establishing the induction. □

From Lemma 7, it is clear that no splitter ever has more than an α fraction of its inputs containing paths to be extended to outputs. Therefore the path extension algorithm never is swamped by placeholders and always works as planned at each level, cutting down the number of stalled paths by a factor of ρ during each phase. Hence, $\lg(M/L)/\lg(1/\rho)$ phases after the wavefront arrives at a splitter of size M, all paths are extended. Thus, the algorithm establishes all paths successfully in

$$\begin{aligned}
&\max_{0\leq i\leq \lg(N/L)}\left(i + \frac{\log\frac{N}{2^iL}}{\log\frac{1}{\rho}} + \frac{\log\frac{N}{L} - i}{C}\right) \\
&= \max_{0\leq i\leq \lg(N/L)}\left(\frac{\log\frac{N}{L}}{\log\frac{1}{\rho}} + \frac{\log\frac{N}{L}}{C} + i\left(1 - \frac{1}{\log\frac{1}{\rho}} - \frac{1}{C}\right)\right) \\
&= \log(N/L)
\end{aligned}$$

phases, provided $\rho < 1/4$. This is because if a path is last stalled at level i, then it passes through level i by phase $i + \log(N/2^iL)/\log(1/\rho)$ and reaches the end $(\log(N/L) - i)/C$ phases later. At first, this result seems too good to be true, but recall that stalled real paths catch up to the wavefront very quickly once they get through, and that they get through at a very high rate. Hence, all real paths get through to the final level along with the wavefront!

By propagating the cancellations from the previous phase at the same time as the paths are extended, a single phase can be implemented in $\max(C, T+1)$ steps. (The extra step in $T+1$ is for initiating placeholders for stalled paths.) By using very good splitters ($\delta \approx 1$), α small, d large, $C = 3$, and $T = 1$, we can obtain a $(2+\varepsilon)\log N$ step algorithm for routing all the paths. It is worth noting that this beats the best previous bounds for store and forward routing [13] by a factor of 2. Unfortunately, d and L need to be quite large to achieve this bound. For more reasonable values of d (< 10) and L (< 300), we can achieve provable routing times of about $100\lg N$. Fortunately, the algorithms appear to work faster in reality. We are currently running computer simulations on the algorithms and hope to soon have some data illustrating this point.

It is also worth noting that each switch needs only to keep track of a few bits of information to make its decisions. This is because only the ith bit of the destination is needed to make a switching decision at level i, and therefore a switch at that level looks at this bit, strips it off, and passes the rest of the destination address onward. The path as a whole snakes forward through the network. If it ever gets blocked, the entire snake halts behind it. The implementation details for this scheme are straightforward. Previously, only the AKS sorting circuit was known to achieve this performance for bounded-degree networks, but at a much greater cost in complexity and constant factors.

4.2 Routing many paths in a nonblocking fashion on a multi-Benes network

It is not difficult to implement the algorithm just described on a multi-Benes network. We need to define the unique-neighbor property for mergers, but it is straightforward. If we have to route around existing paths, however, then we have to combine the algorithm with the kind of analysis used in Section 3. In particular, we need to modify the definition of being blocked so that a node on level l is *blocked* if more than $2\beta - d - 1$ of its up (or down) neighbors on level $l+1$ are busy or blocked. (As before, we assume that $\beta > d/2$.) *Working* nodes will then be guaranteed to have at least $2d - 2\beta + 1$ working neighbors. Since every set of size

k is linked to at most $(2\beta - d - 1)k$ busy or blocked up (or down) nodes at the next level, as well as $(2\beta - d)k$ unique-neighbors (by lemma 4), therefore at least k of the unique-neighbors must still be working. Hence, any set of $k \leq \alpha M$ working inputs in an M-input splitter will have a $(\alpha, 1/d)$ unique-neighbor property, which is sufficient for the routing algorithm to work.

Of course, we must also check that the new blocking definition does not result in any inputs to the multi-Benes network becoming blocked. This can be done with an argument similar to that in Lemma 1. Roughly speaking, if an α fraction of the inputs to an M-input splitter were to become blocked, by upper inputs (say), then each of the inputs is incident to $2\beta - d - 1$ blocked or busy upper inputs. Hence, of the $\alpha\beta M$ upper output neighbors, at most $(2d - 2\beta + 1)\alpha M$ of them can be working. This means that at least a $2(3\beta - 2d - 1)\alpha$ fraction of the upper outputs are busy or blocked. By induction, however, the fraction of the blocked upper outputs is at most α and thus

$$2(3\beta - 2d - 1)\alpha \leq \alpha + \frac{1}{L}$$

which is a contradiction if $L > 1/\alpha(6\beta - 4d - 3)$. Of course, for this argument to work, we need d large enough so that it is possible for $\beta > (4d + 3)/6$. It is likely that this can be improved to $\beta > d/2$, but the details will be left to the final paper.

5 Multiparty calls and multiple callers

If all the parties of a multiparty call are known to a caller at the start of the call, it is easy to extend the algorithms in Sections 3 and 4 to route the call using the greedy algorithm. The path simply creates branches where necessary in levels $\lg N$ through $2 \lg N$ of the multi-Benes network to reach all the desired output terminals. The bit complexity of the algorithm may increase, however, to reflect the Kolmogorov complexity of the set of outputs that the path must reach.

If parties to a multiparty call are to be added after the call is already underway, then we use a multi-Benes network with wraparound connections, and add parties by routing paths from parties already involved in the call. Each addition takes $O(\log N)$ steps, and by being careful, we can insure that the resulting "depth" of the call is at most $O(\log^2 N)$. This is not quite as elegant as the solutions proposed in [9] for generalized non-blocking networks, but no routing algorithms are known at all for those constructions.

If many parties want to call the same output terminal, then we have two options: merging the callers into a single multiparty call, or giving busy signals to all but one of the callers. Both options can be performed in $O(\log N)$ bit-steps, but both require the use of the AKS sorting circuit for doing some processing. In particular, we use the AKS circuit to sort the calls according to their destination, and then use a prefix operation to combine them or to send busy signals to redundant calls. Calls not receiving busy signals can then proceed as before. Calls that are combined are handled in a manner analogous to the way we handled multiparty calls, only in reverse.

The details of all of these procedures will be included in the final draft of the paper.

6 Acknowledgments

The authors thank Nick Pippenger for many helpful comments and for suggesting that a nonblocking network might be constructed by treating busy switches as if they were faulty.

References

[1] W. Aiello, T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. Unpublished manuscript.

[2] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(N \log N)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 1–9, April 1983.

[3] L. A. Bassalygo and M. S. Pinsker. Complexity of optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.

[4] L. A. Bassalygo and M. S. Pinsker. Asymptotically optimal networks for generalized rearrangeable switching and generalized switching without rearrangement. *Problemy Peredachi Informatsii*, 16:94–98, 1980.

[5] V. E. Benes. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1641–1656, July 1964.

[6] D. G. Cantor. On non-blocking switching networks. *Networks*, 1:367–377, 1971.

[7] D. Dolev, C. Dwork, N. Pippenger, and A. Widgerson. Superconcentrators, generalizers and generalized connectors with limited depth. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 42–51, April 1983.

[8] P. Feldman, J. Friedman, and N. Pippenger. Nonblocking networks. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 247–254, May 1986.

[9] P. Feldman, J. Friedman, and N. Pippenger. Wide-sense nonblocking networks. *SIAM Journal of Discrete Mathematics*, 1(2):158–173, May 1988.

[10] J. Friedman. A lower bound on strictly nonblocking networks. *Combinatorica*, 8(2):185–188, 1988.

[11] C. P. Kruskal and M. Snir. A unified theory of interconnection network structure. *Theoretical Computer Science*, 48:75–94, 1986.

[12] F. T. Leighton. Parallel computation using meshes of trees. In *1983 Workshop on Graph-Theoretic Concepts in Computer Science*, pages 200–218, Linz, 1984. Trauner Verlag.

[13] T. Leighton and B. Maggs. Expanders might be practical: fast algorithms for routing around faults in multibutterflies. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 384–389. IEEE, October 1989.

[14] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C–34(10):892–901, October 1985.

[15] G. A. Margulis. Explicit constructions of concentrators. *Problems of Information Transmission*, 9:325–332, 1973.

[16] G. M. Masson and B. W. Jordan, Jr. Generalized multi-stage connection networks. *Networks*, 2:191–209, 1972.

[17] D. Nassimi and S. Sahni. Parallel permutation and sorting algorithms and a new generalized connection network. *Journal of the ACM*, 29(3):642–667, July 1982.

[18] Yu. P. Ofman. A universal automaton. *Transactions of the Moscow Mathematical Society*, 14:186–199, 1965.

[19] N. Pippenger. Personal communication.

[20] N. Pippenger. *The complexity theory of switching networks.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1973.

[21] N. Pippenger. On rearrangeable and nonblocking switching networks. *Journal of Computer and System Sciences*, 17:145–162, 1978.

[22] N. Pippenger. Telephone switching networks. In *Proceedings of Symposia in Applied Mathematics*, volume 26, pages 101–133, 1982.

[23] N. Pippenger and L. G. Valiant. Shifting graphs and their applications. *Journal of the ACM*, 23:423–432, 1976.

[24] N. Pippenger and A. C.-C. Yao. Rearrangeable networks with limited depth. *SIAM Journal of Algebraic and Discrete Methods*, 3:411–417, 1982.

[25] C. E. Shannon. Memory requirements in a telephone exchange. *Bell System Technical Journal*, 29:343–349, 1950.

[26] J. S. Turner. Practical wide-sense nonblocking generalized connectors. Technical Report WUCS-88-29, Department of Computer Science, Washington University, St. Louis, MO, 1989.

[27] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 241–250, May 1989.